



Using Automatic Differentiation in Newton-Krylov Methods

Rootfinders' Ball
Livermore, CA
Aug 6—8, 2003



Outline

- Motivation
- Integrating AD with PETSc
- Experimental results
- What's next?

Thanks to Boyana Norris, the PETSc team (especially Lois McInnes and Barry Smith), and the MICS subprogram of the Office of Advanced Scientific Computing Research, U.S. Department of Energy

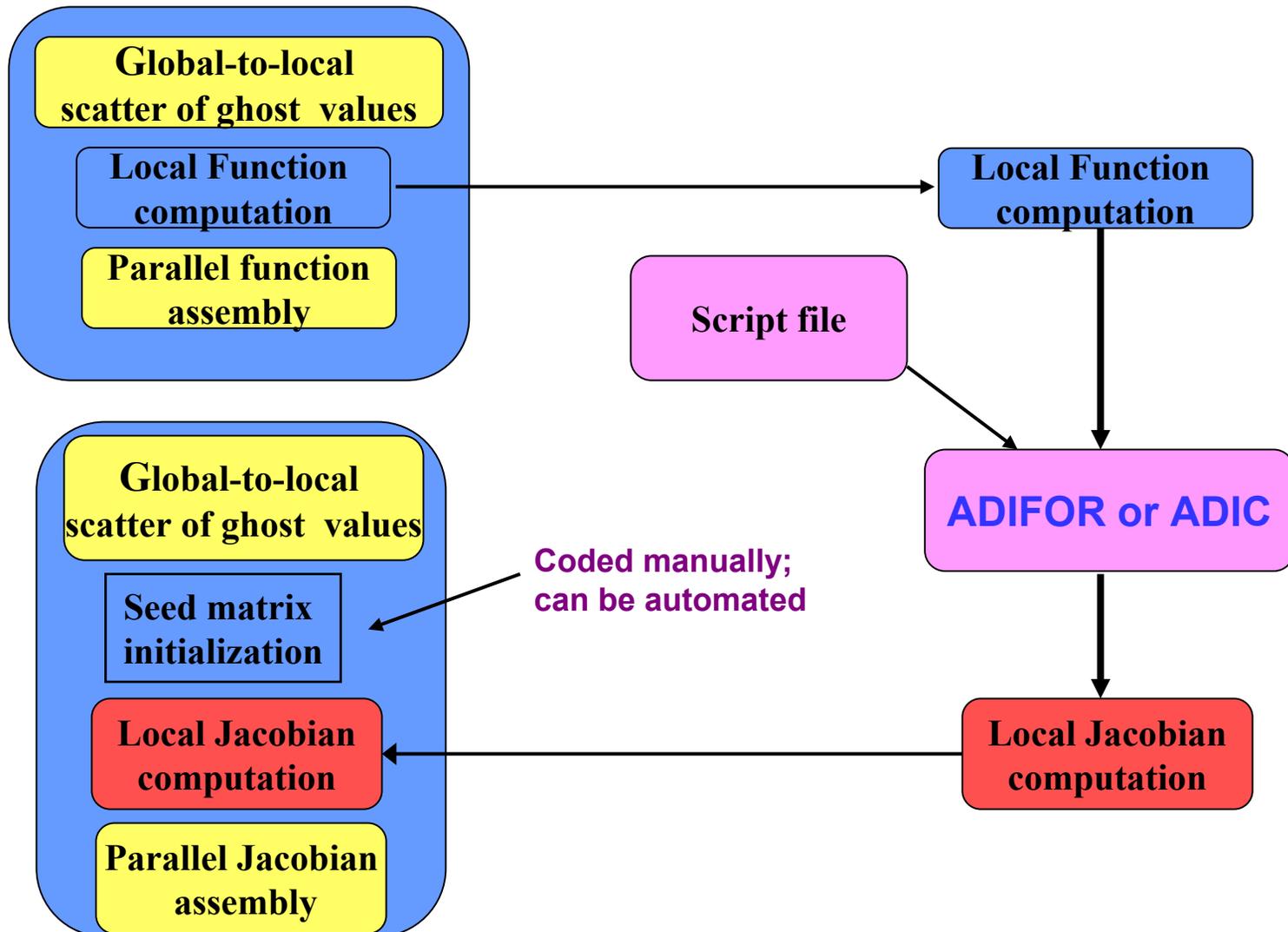
Motivation

- Automatic Differentiation
 - Analytic derivatives of source code
 - Cheap Jv
 - Cheap $J^T v$ (reverse mode)
- Newton-Krylov requires:
 - J [preconditioning]
 - Jv [Krylov]
 - $J^T v$ [BCG, QMR]
- When do analytic derivatives pay off?

Automatic Differentiation

- Technique for augmenting code for computing a function with code for computing derivatives
- Analytic differentiation of elementary operations/functions, propagation by chain rule
- Can be implemented using source transformation or operator overloading
- Two main modes
 - Forward: propagates derivatives from independent to dependent variables
 - Reverse (adjoint): propagates derivatives from dependent to independent variables

Using AD with PETSc (old way)



Automating AD (User's Perspective)

- User provides subdomain function (FormLocalFunction)
- Using AD requires:
 - Changing: `DMMGSetSNES(dmmg,FormFunction,0)` call to `DMMGSetSNESLocal(dmmg,FormFunctionLocal,0, ad_FormFunctionLocal, admf_FormFunctionLocal)`
 - Adding a comment of the form `/* Process adiC: FormFunctionLocal */`
 - Switch at runtime between AD and FD using the options `-dmmg_jacobian_ad` and `-dmmg_jacobian_fd` (equivalent options for matrix free methods exist)

PETSc Example



```
int FormFunctionLocal (DALocalInfo *info, PetscScalar **x, PetscScalar **f, AppCtx *user)
{
    int          ierr, i, j;
    PetscReal    two = 2.0, lambda, hx, hy, hxdhy, hydhx, sc;
    PetscScalar  u, uxx, uyy;

    PetscFunctionBegin;

    lambda = user->param;
    hx     = 1.0 / (PetscReal) (info->mx-1);
    hy     = 1.0 / (PetscReal) (info->my-1);
    sc     = hx*hy*lambda;
    hxdhy  = hx/hy;
    hydhx  = hy/hx;

    /* Compute function over the locally owned part of the grid */
    for (j=info->ys; j<info->ys+info->ym; j++) {
        for (i=info->xs; i<info->xs+info->xm; i++) {
            if (i == 0 || j == 0 || i == info->mx-1 || j == info->my-1) {
                f[j][i] = x[j][i];
            } else {
                u      = x[j][i];
                uxx    = (two*u - x[j][i-1] - x[j][i+1])*hydhx;
                uyy    = (two*u - x[j-1][i] - x[j+1][i])*hxdhy;
                f[j][i] = uxx + uyy - sc*PetscExpScalar(u);
            }
        }
    }

    ierr = PetscLogFlops(11*info->ym*info->xm); CHKERRQ(ierr);
    PetscFunctionReturn(0);
}
```

```

int main(int argc, char **argv)
{
    SNES                snes;                /* nonlinear solver */
    Vec                 x,r;                /* solution, residual vectors */
    Mat                 A,J;                /* Jacobian matrix */
    AppCtx              user;                /* user-defined work context */
    /* ... More variable declarations, initializations ... */

    SNESCreate(PETSC_COMM_WORLD, SNES_NONLINEAR_EQUATIONS, &snes);
    DACreate2d(PETSC_COMM_WORLD, DA_NONPERIODIC, DA_STENCIL_STAR, -4, -4,
               PETSC_DECIDE, PETSC_DECIDE, 1, 1, PETSC_NULL, PETSC_NULL, &user.da);
    /* Extract global vectors from DA; then duplicate for remaining vectors */
    DACreateGlobalVector(user.da, &x); VecDuplicate(x, &r);
    /* Set local function evaluation routine */
    DASETLocalFunction(user.da, (DALocalFunction1) FormFunctionLocal);
    DASETLocalJacobian(user.da, (DALocalFunction1) FormJacobianLocal);
    DASETLocalAdicFunction(user.da, ad_FormFunctionLocal);

    /* ... Decide which function to use based on runtime options ... */

    /* Create matrix data structure; set Jacobian evaluation routine */
    DAGetMatrix(user.da, MATMPIAIJ, &J); A = J;
    DAGetColoring(user.da, IS_COLORING_GHOSTED, &iscoloring);
    MatSetColoring(J, iscoloring);
    ISColoringDestroy(iscoloring);

    SNESSetJacobian(snes, A, J, SNESDAComputeJacobianWithAdic, &user);

    SNESSetFromOptions(snes);
    FormInitialGuess(&user, x);
    SNESsolve(snes, x, &its);

    /* ... Free work space ... */

```

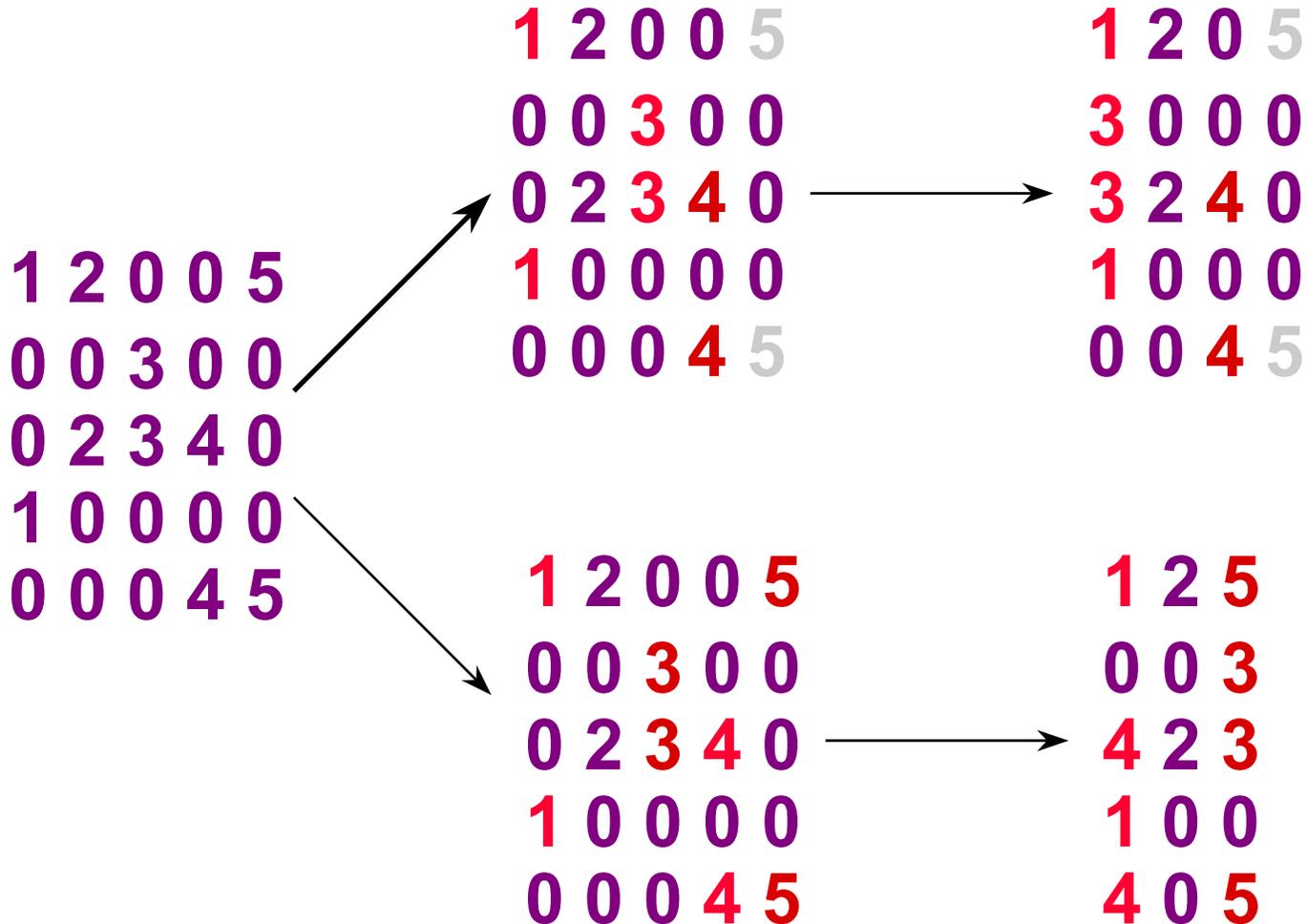
Automating AD (behind the scenes)

- PETSc augmented to automatically allocate several derivative objects
- PETSc extended to support initialization of the so-called “seed matrix” (using CPR coloring or vector)
- ADIC modified to support extraction of a row of the compressed Jacobian
- PETSc support added for assembling matrix from these rows
- PETSc build procedure modified to automatically extract subdomain function and apply ADIC

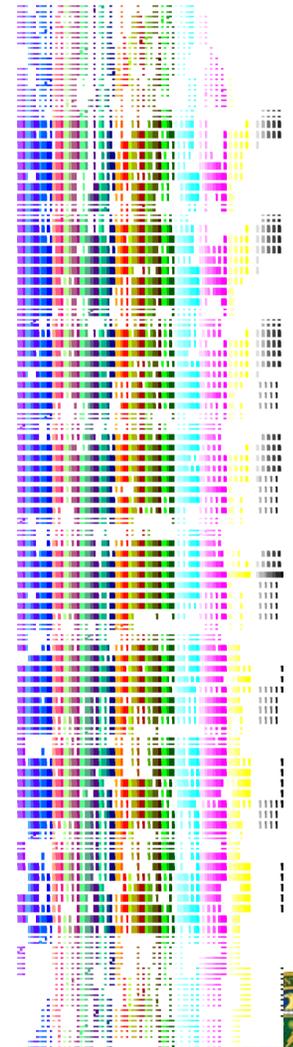
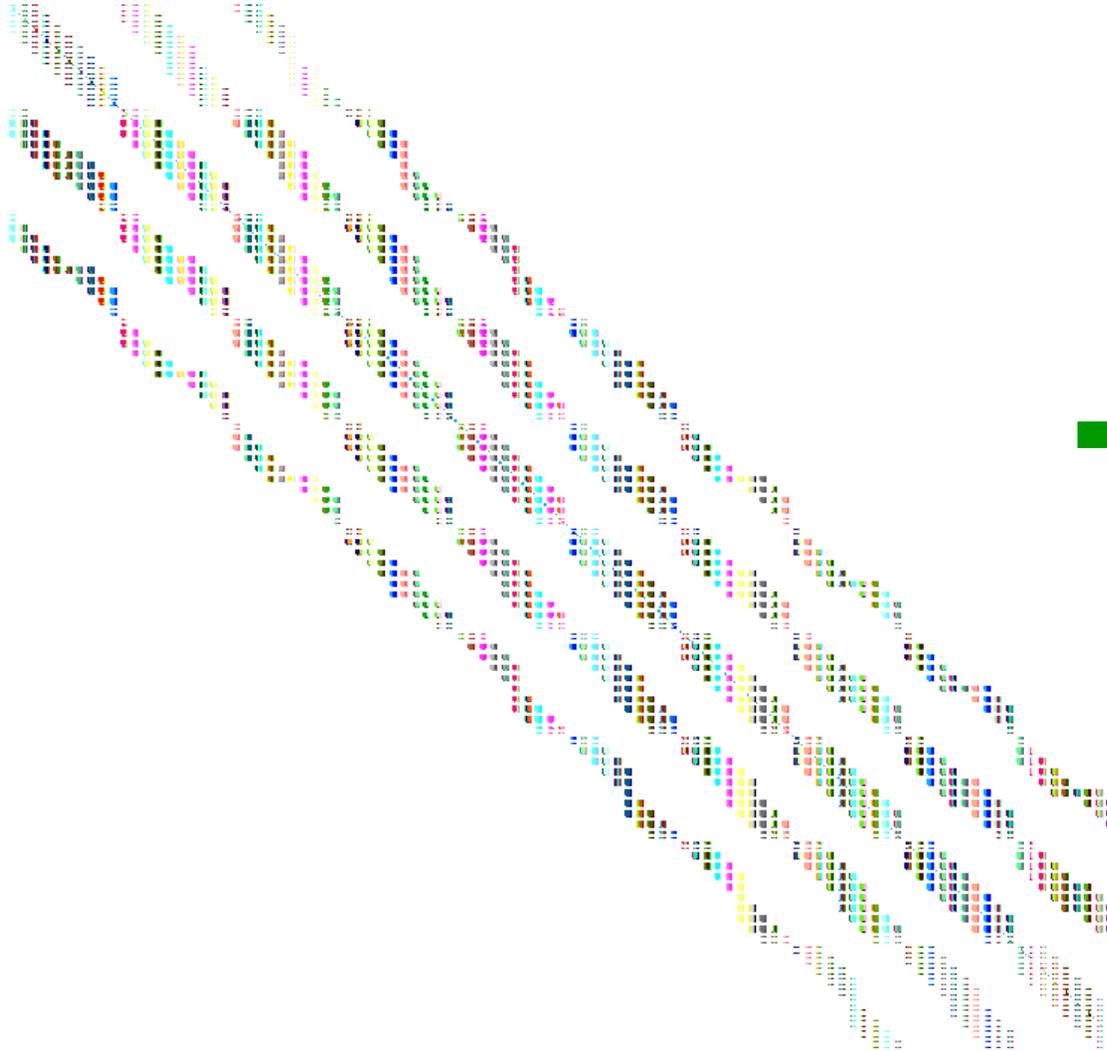
Matrix Coloring

- Jacobian matrices are often sparse
- The forward mode of AD computes $J \times S$, where S is usually an identity matrix or a vector
- Can “compress” Jacobian by choosing S such that **structurally orthogonal** columns are combined
- A set of columns are structurally orthogonal if no two of them have nonzeros in the same row
- Equivalent problem: color the graph whose adjacency matrix is $J^T J$

Matrix Coloring



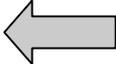
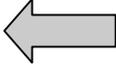
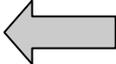
Compressed Jacobian



Hybrid AD/FD for Jacobian-vector Products

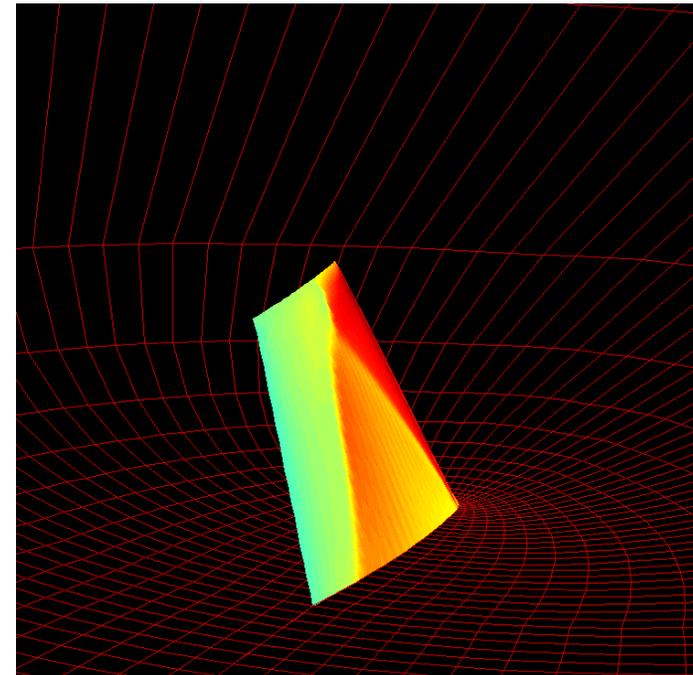
- FD
 - $F'(x) v \approx [F(x+hv) - F(x)]/h$
 - costs approximately 1 function evaluation
 - challenges in computing the differencing parameter, h , since we must balance truncation and round-off errors
- AD
 - costs approximately 2 function evaluations
 - no difficulties in parameter estimation
- Hybrid FD/AD
 - switch from FD to AD when $\|F\| / \|F_0\| < \delta$

Another Hybrid Strategy

- Reference: K. Turner and H. Walker, “Efficient High Accuracy Solutions with GMRES(m)”, SISC, May 1992
- Efficient High Accuracy Algorithm
Let an initial x be given. Until termination, do:
 - Compute $r(x) = b - Ax$ accurately.  Use AD
 - Compute z by one cycle of GMRES(m).  Use FD
 - Update $x = x + z$ accurately and compute $r(x) = b - Ax$ accurately.  Use AD

PETSc Applications

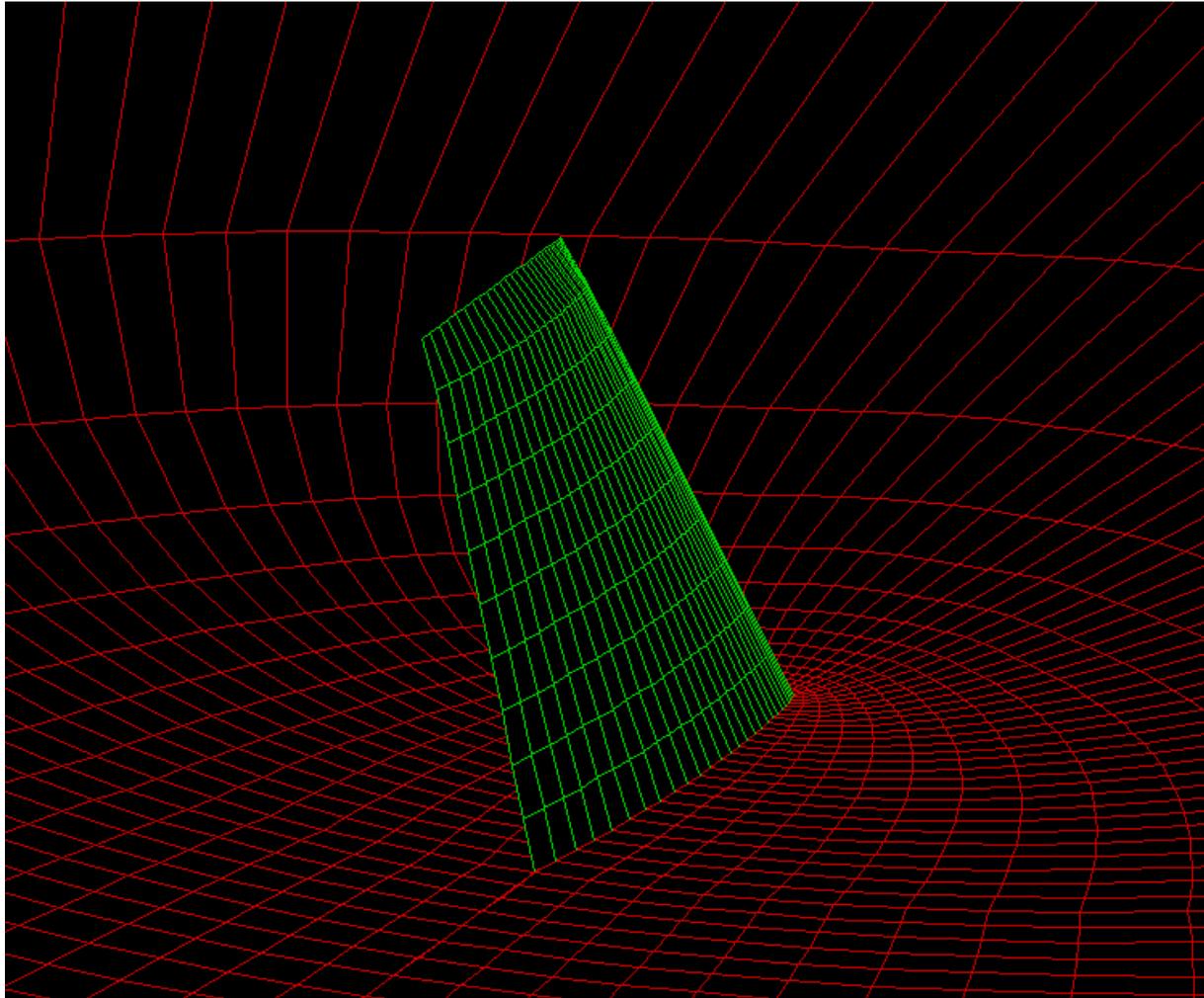
- Toy problems
 - Solid fuel ignition: finite difference discretization; Fortran & C variants; differentiated using ADIFOR, ADIC
 - Driven cavity: finite difference discretization; C implementation; differentiated using ADIC
- Euler code
 - Based on legacy F77 code from D. Whitfield (MSU)
 - Finite volume discretization
 - Up to 1,121,320 unknowns
 - Mapped C-H grid
 - Fully implicit steady-state
 - Transonic flow over ONERA M6 wing
 - Newton w/ pseudo-transient cont.
 - Lagged, approximate Jacobian precondition.
 - RASM/GMRES



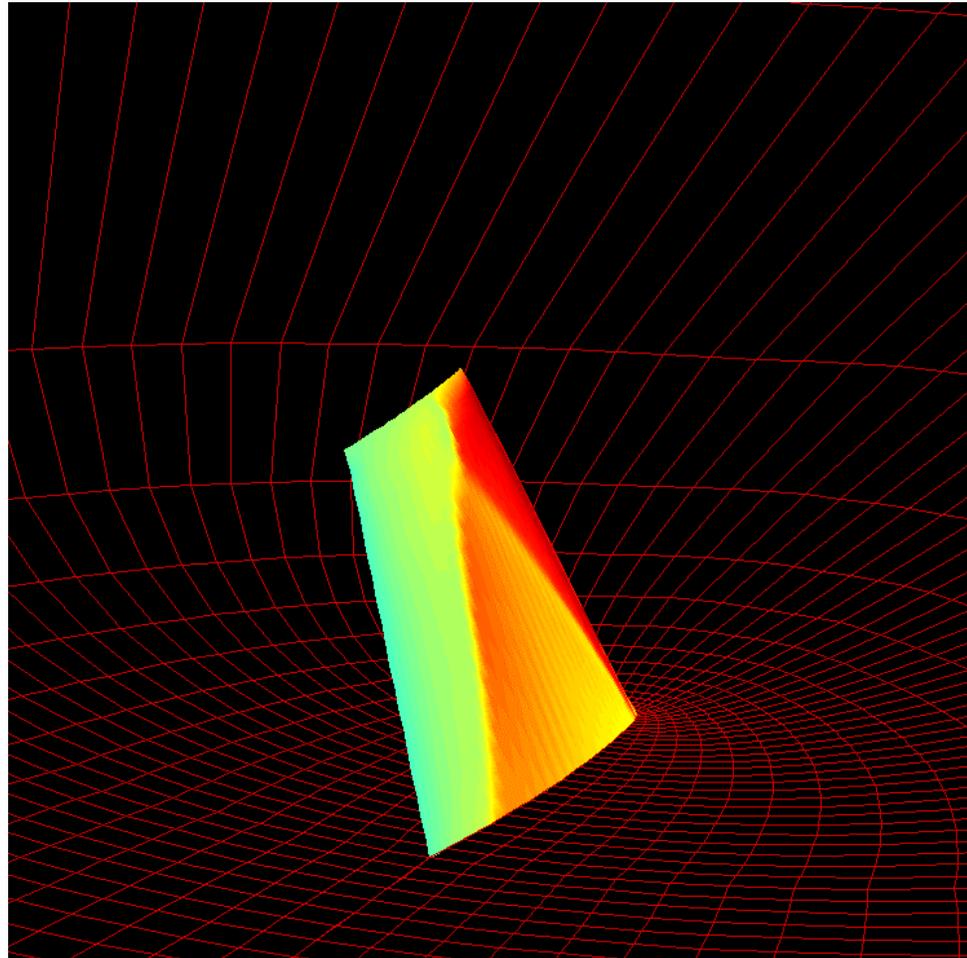
Euler Problem Parameters

- Transonic flow over ONERA M6 wing, at 3.06° angle of attack (exhibits κ -shock)
- 98x18x18 mesh
 - Jacobian has 158,760 rows, 4,636,200 nonzeros
- Use Newton's method with pseudo-transient continuation
- Use a lagged, approximate Jacobian for preconditioning (compute using FD every 10 iterations - in principle, could use AD)
- Solve linear systems using GMRES with Restricted Additive Schwarz preconditioning

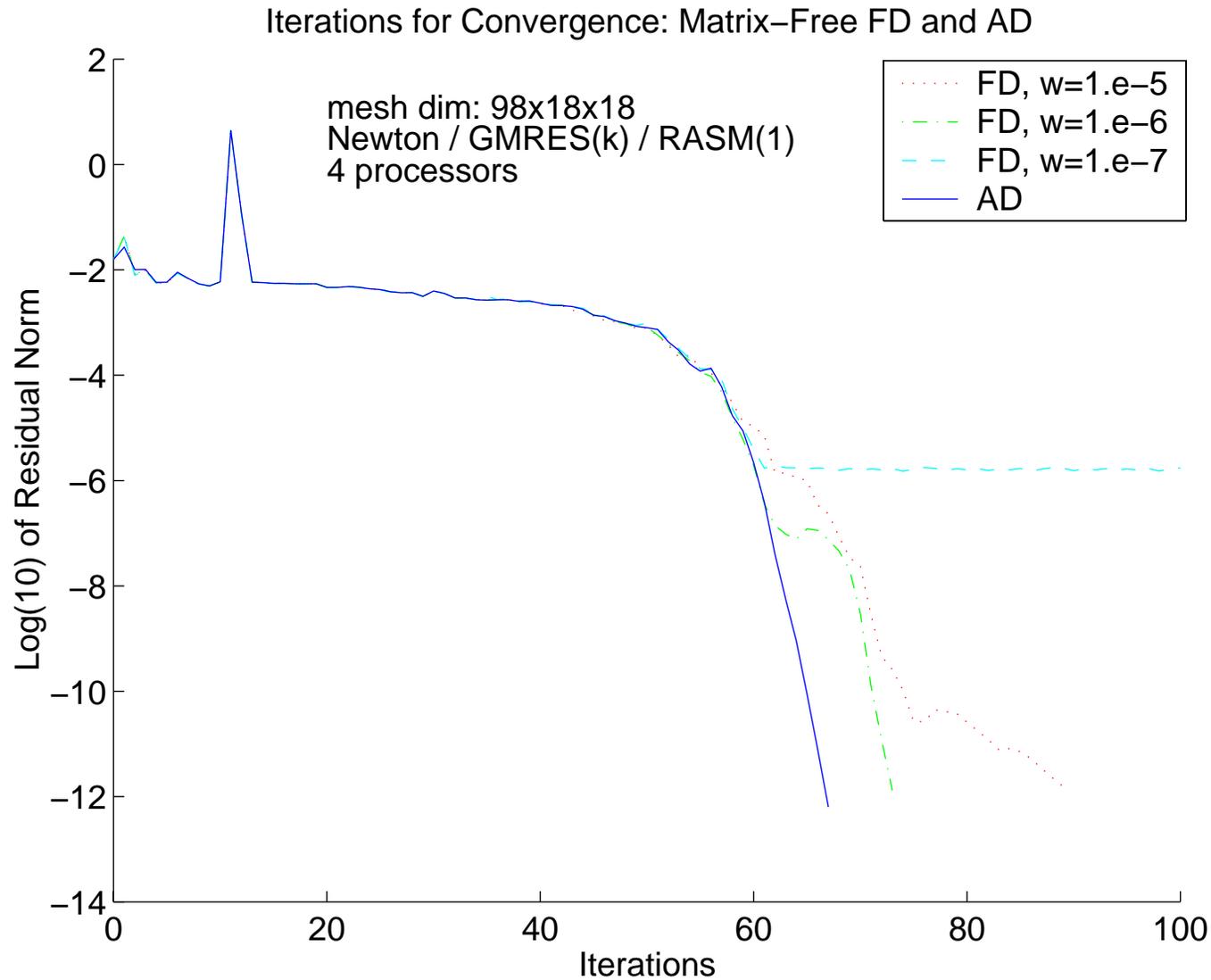
C-H Structured Grid



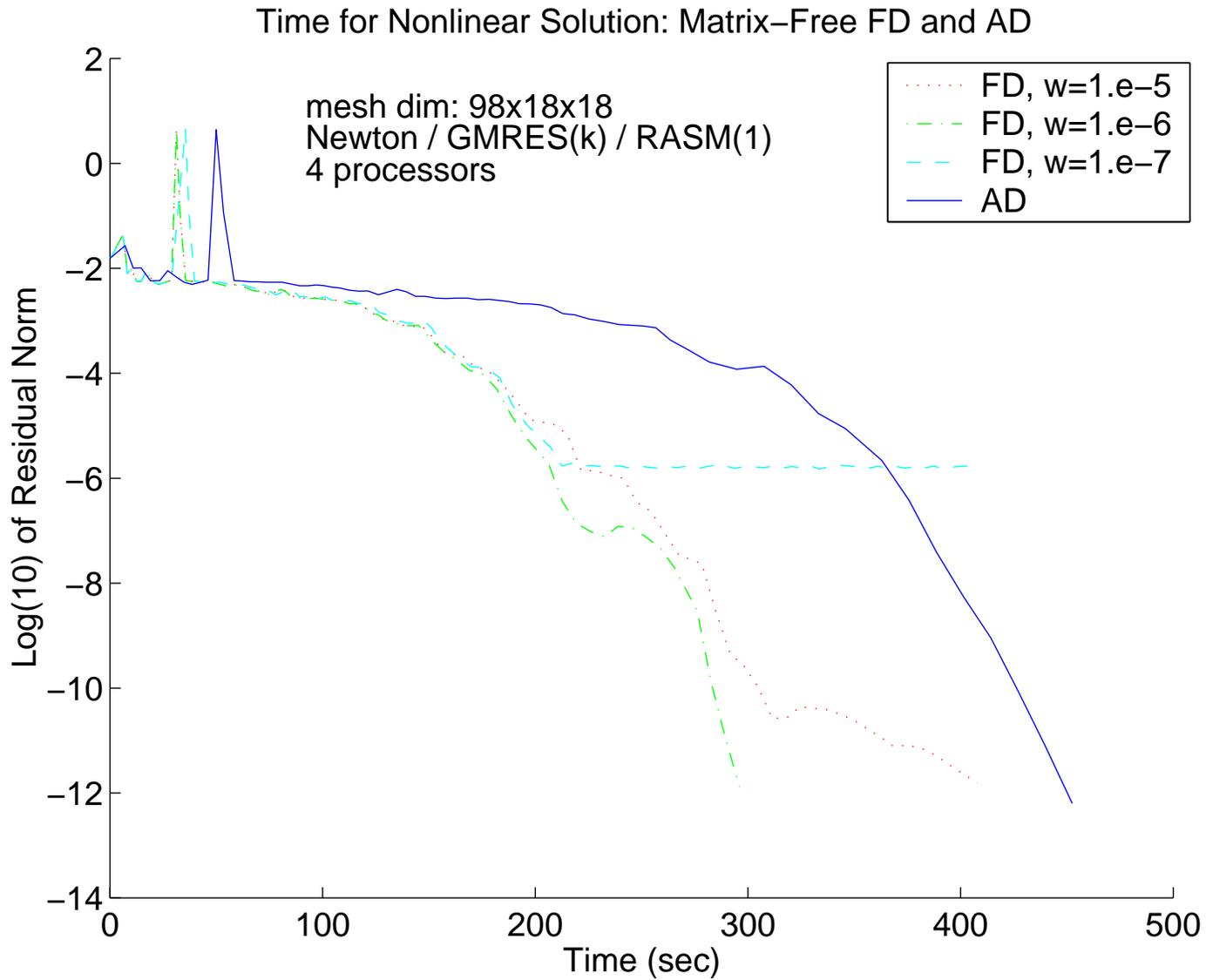
Mach Contours



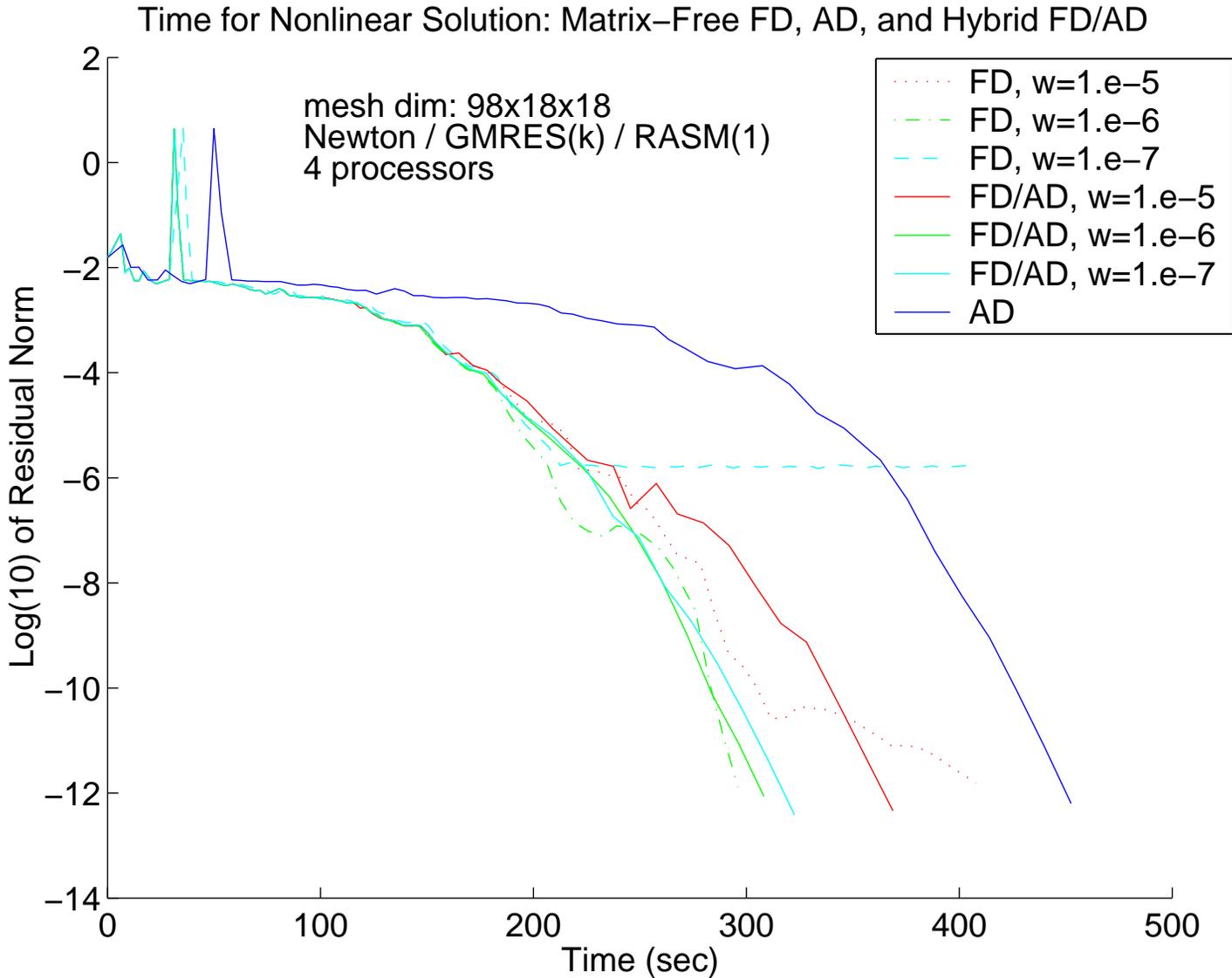
Euler: AD vs. FD: Iterations



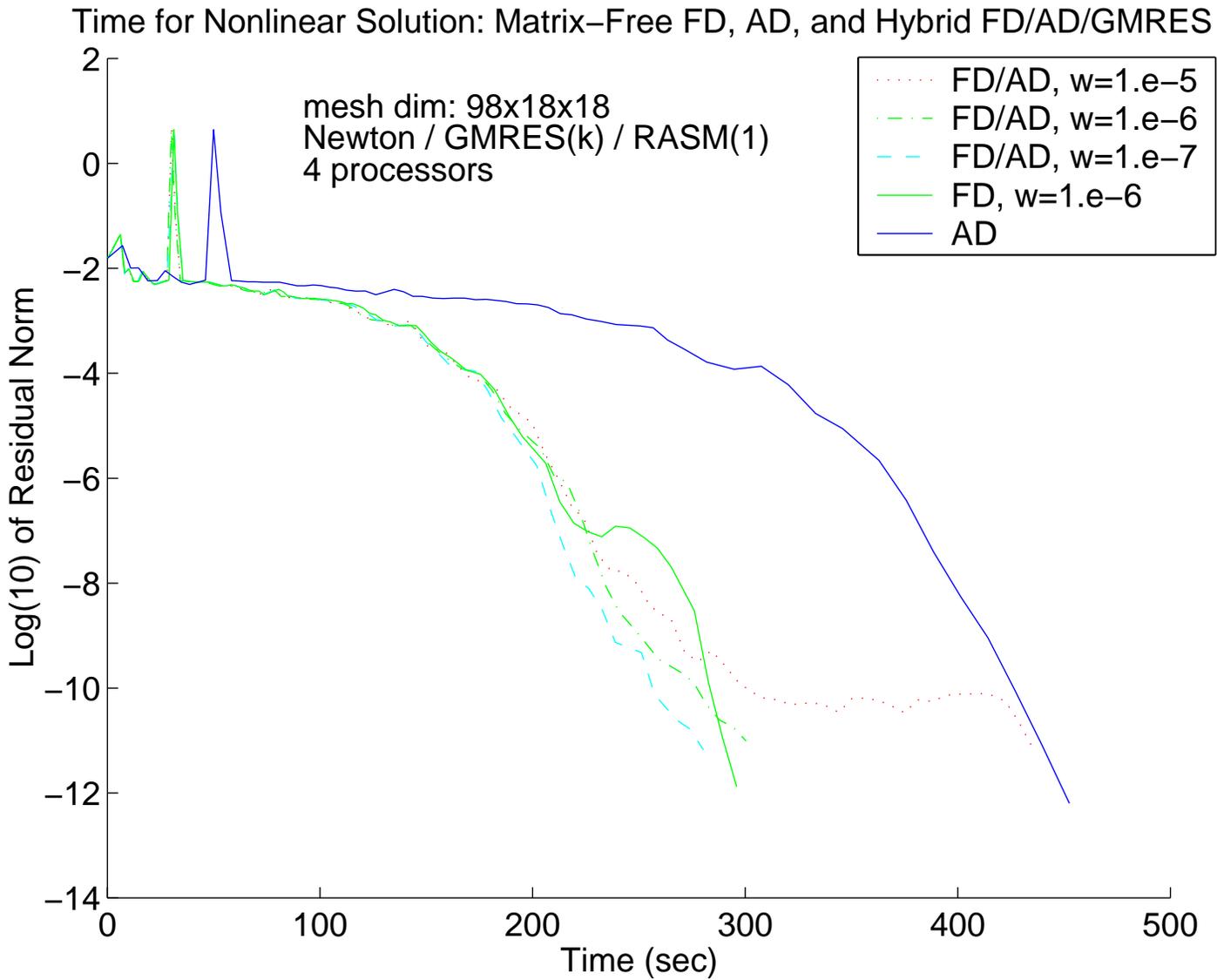
Euler: AD vs. FD: Time



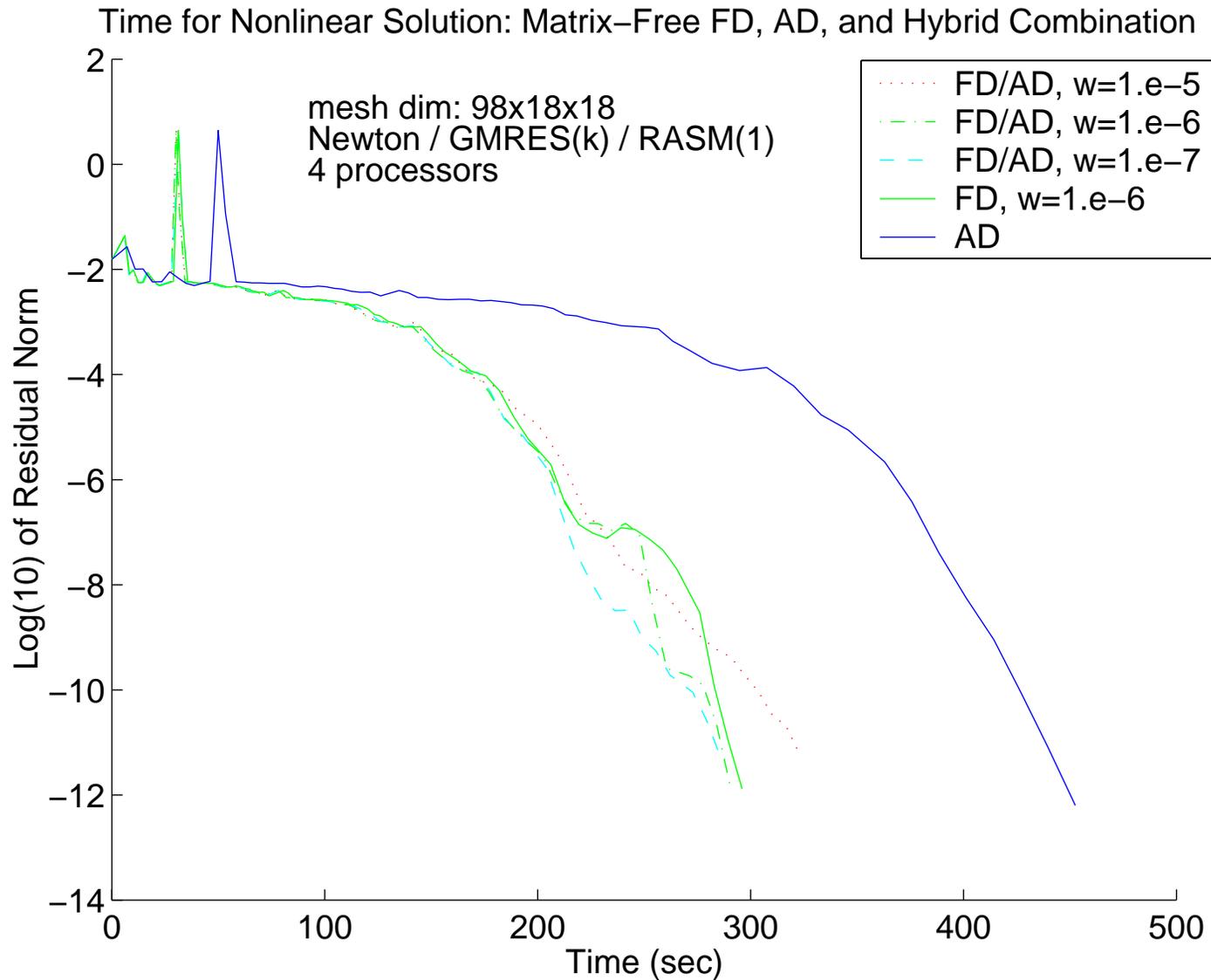
Hybrid AD/FD Method (switching)



Hybrid AD/FD Method (Turner-Walker)



Hybrid AD/FD Method (Combination)

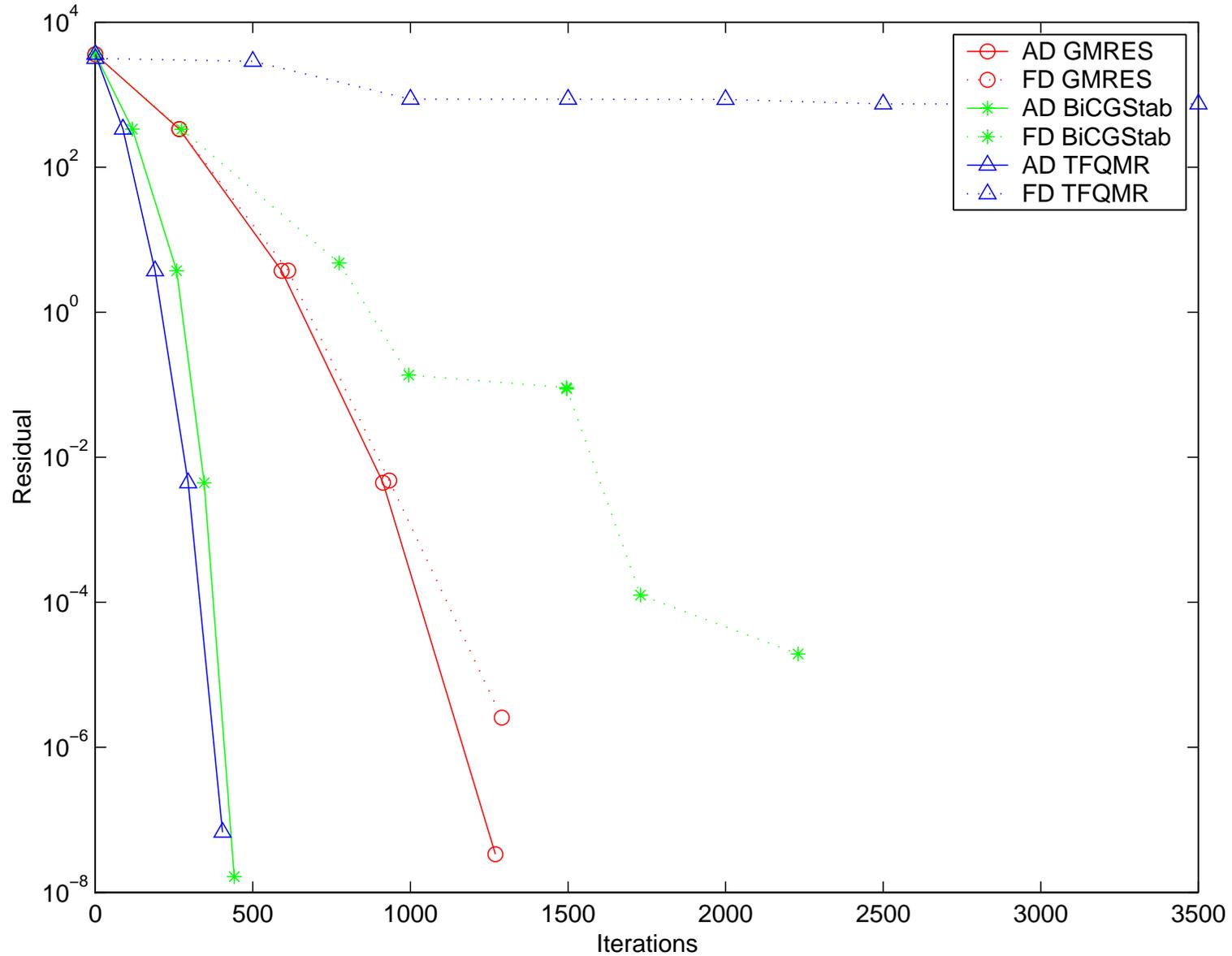


Driven Cavity

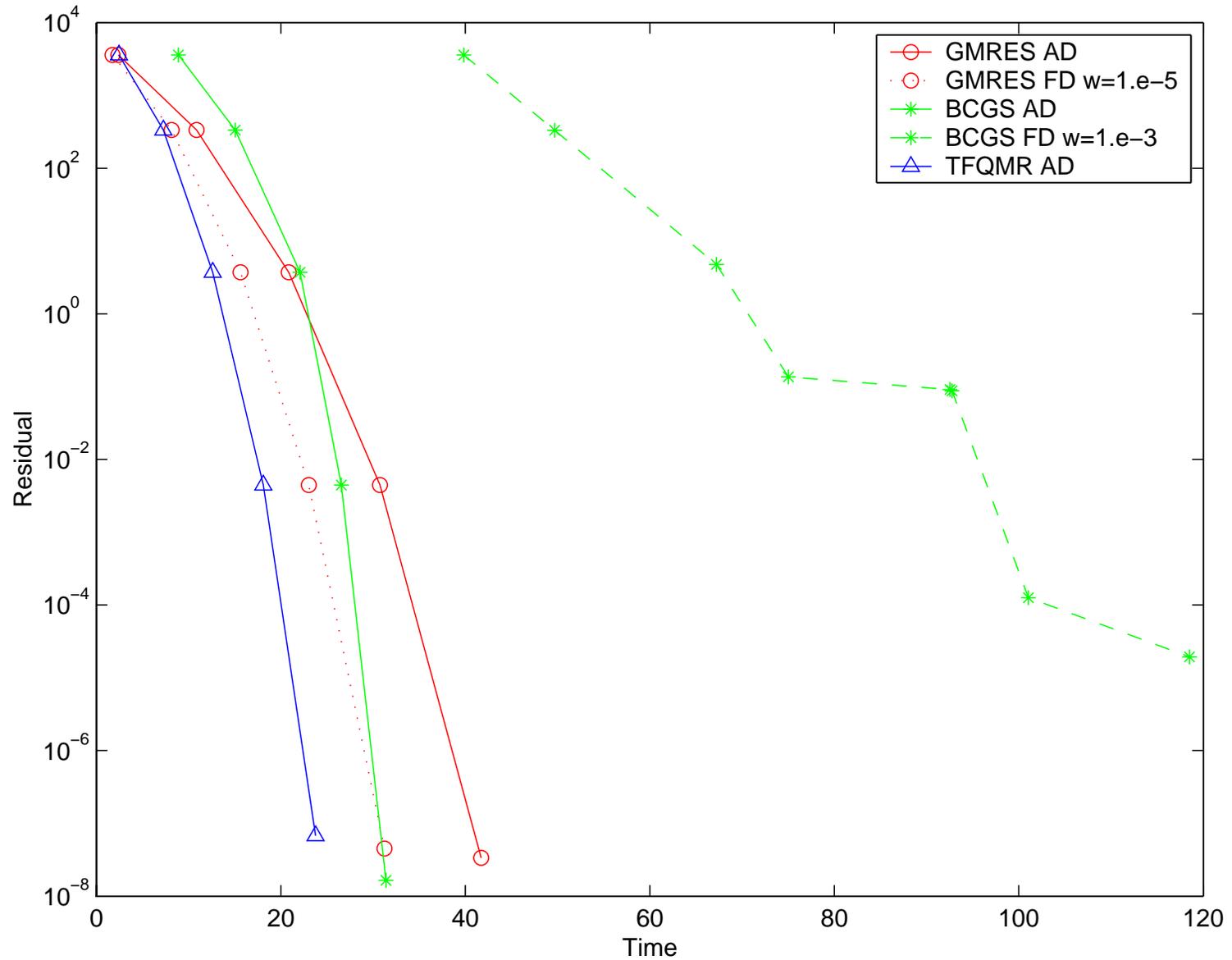
- Problem formulation:
 - Grashof number 10^6 , lid velocity 100 (Coffey, Kelley, Keyes)
 - Grid sequencing with 5 levels
 - 96×96 grid on finest level, 6×6 on coarsest level
- Solution techniques
 - Compute the full Jacobian for use in an ILU-2 preconditioner and apply the Jacobian matrix-free in a Krylov solver
 - Apply the Jacobian in a completely matrix-free manner using a multigrid preconditioner (w/ Krylov smoother) and FGMRES solver
 - Compute the full Jacobian and use a full multigrid solver, with zero, two, or three iterations of a Krylov accelerator

```
ex19 -grashof 100000 -lidvelocity 100 -da_grid_x 6 -da_grid_y 6 -dmmg_nlevels 5 \  
-dmmg_grid_sequence -ksp_type <type> -pc_type ilu -pc_ilu_levels 2 -snes_mf_err <err> \  
-snes_monitor -ksp_monitor -ksp_max_it 500 -dmmg_jacobian_mf_[ad|fd]_operator
```

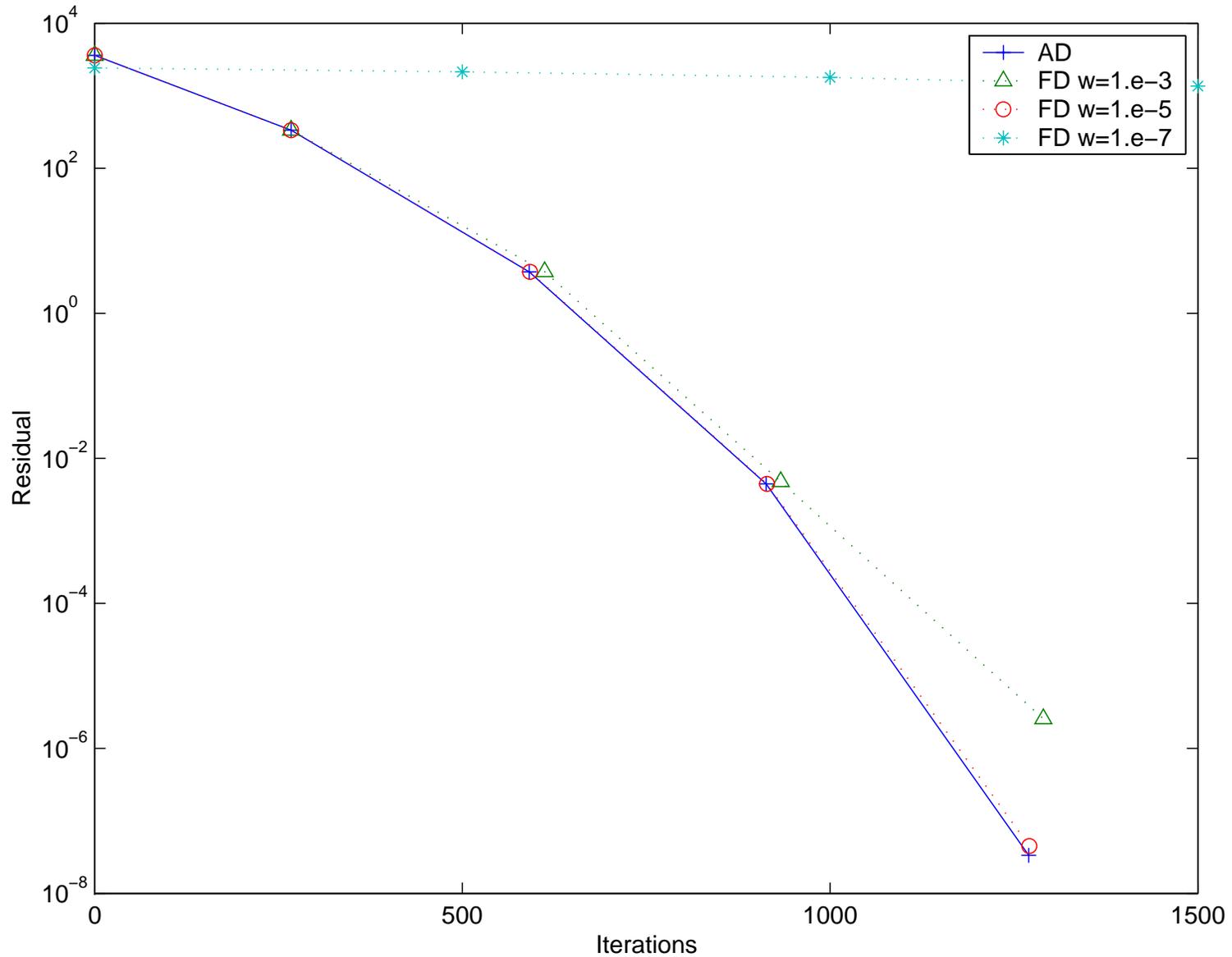
Full/MF ILU-2 Preconditioner: Iterations



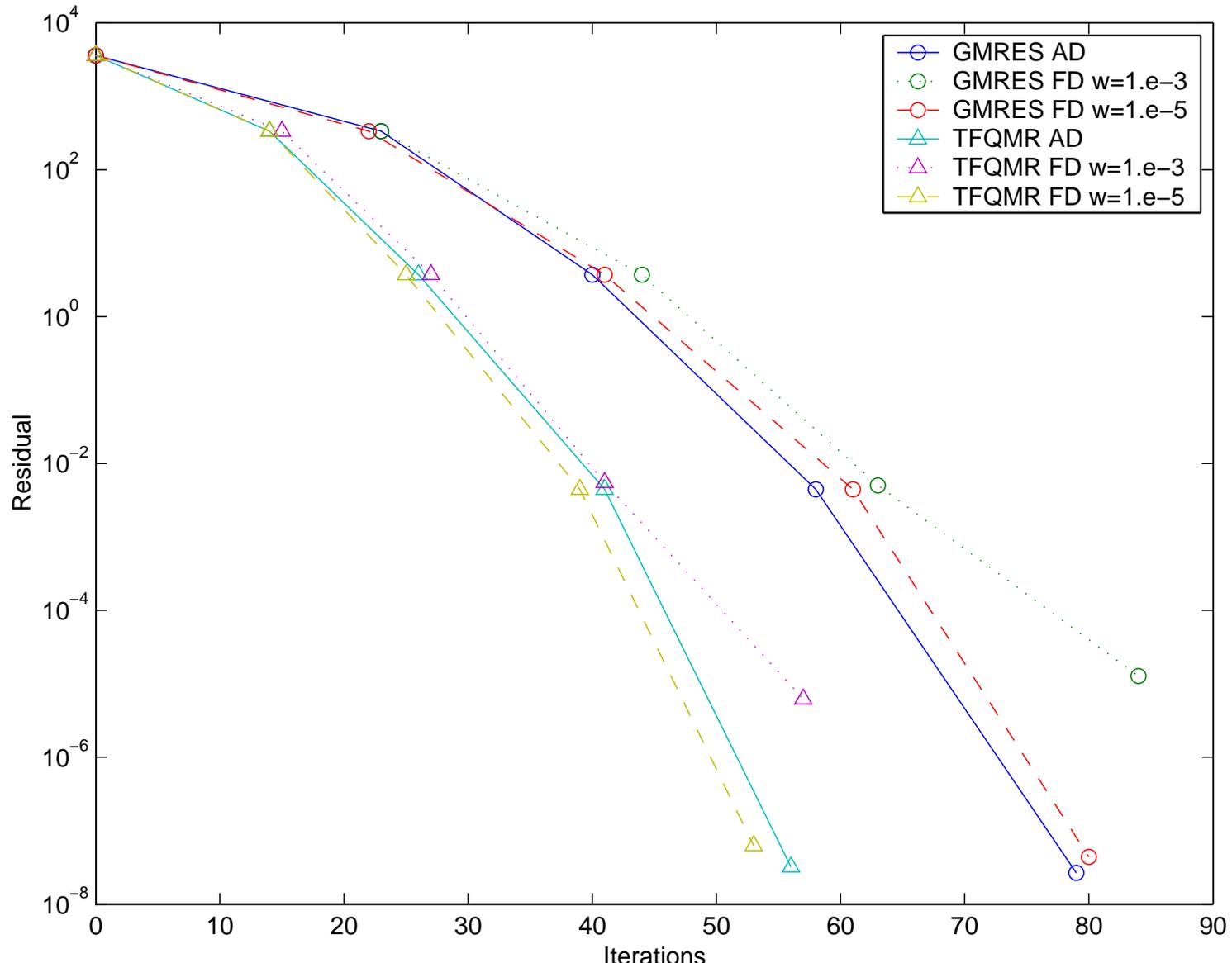
Full/MF ILU-2 Preconditioner: Time



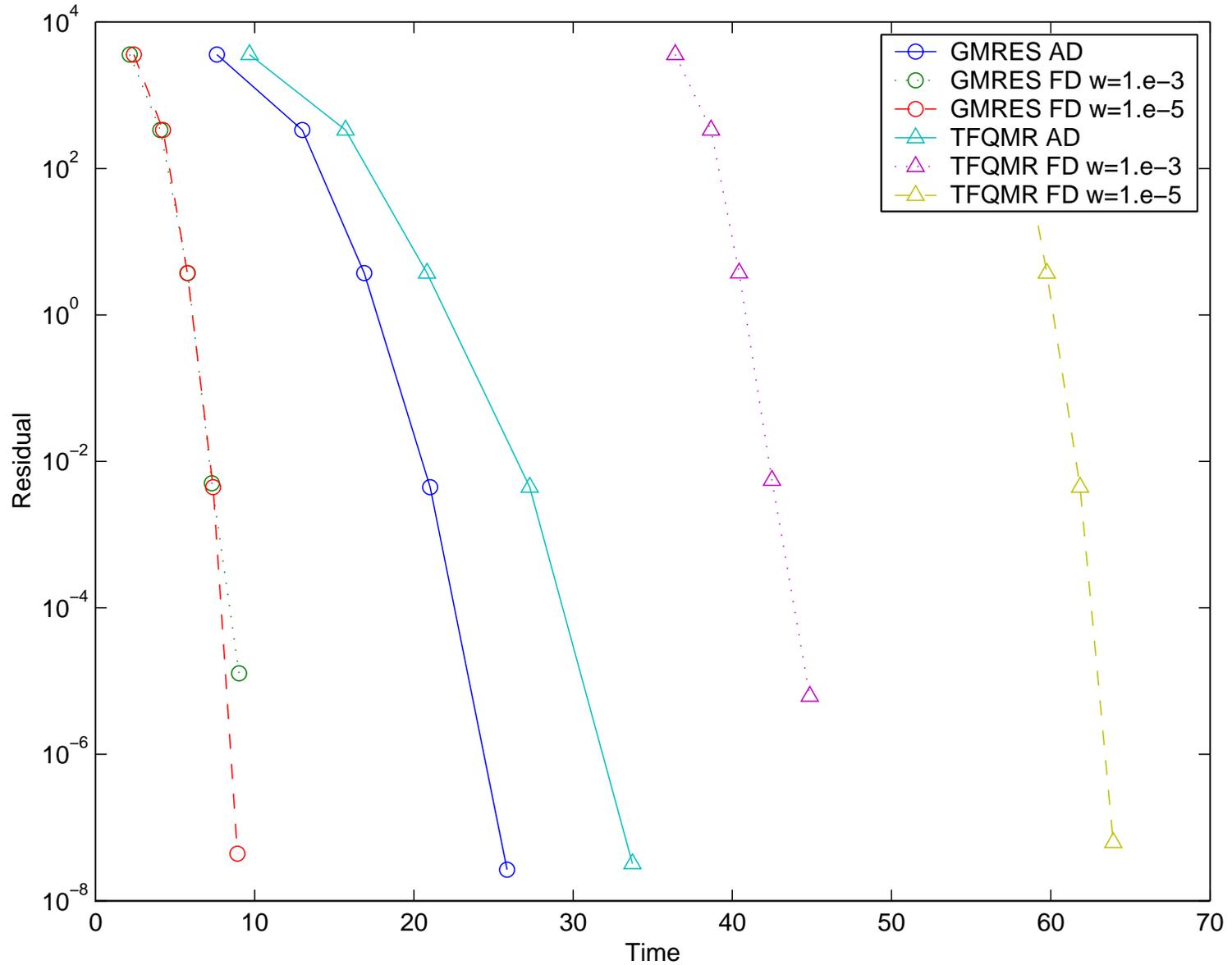
Full/MF ILU-2 Preconditioner: Stepsize



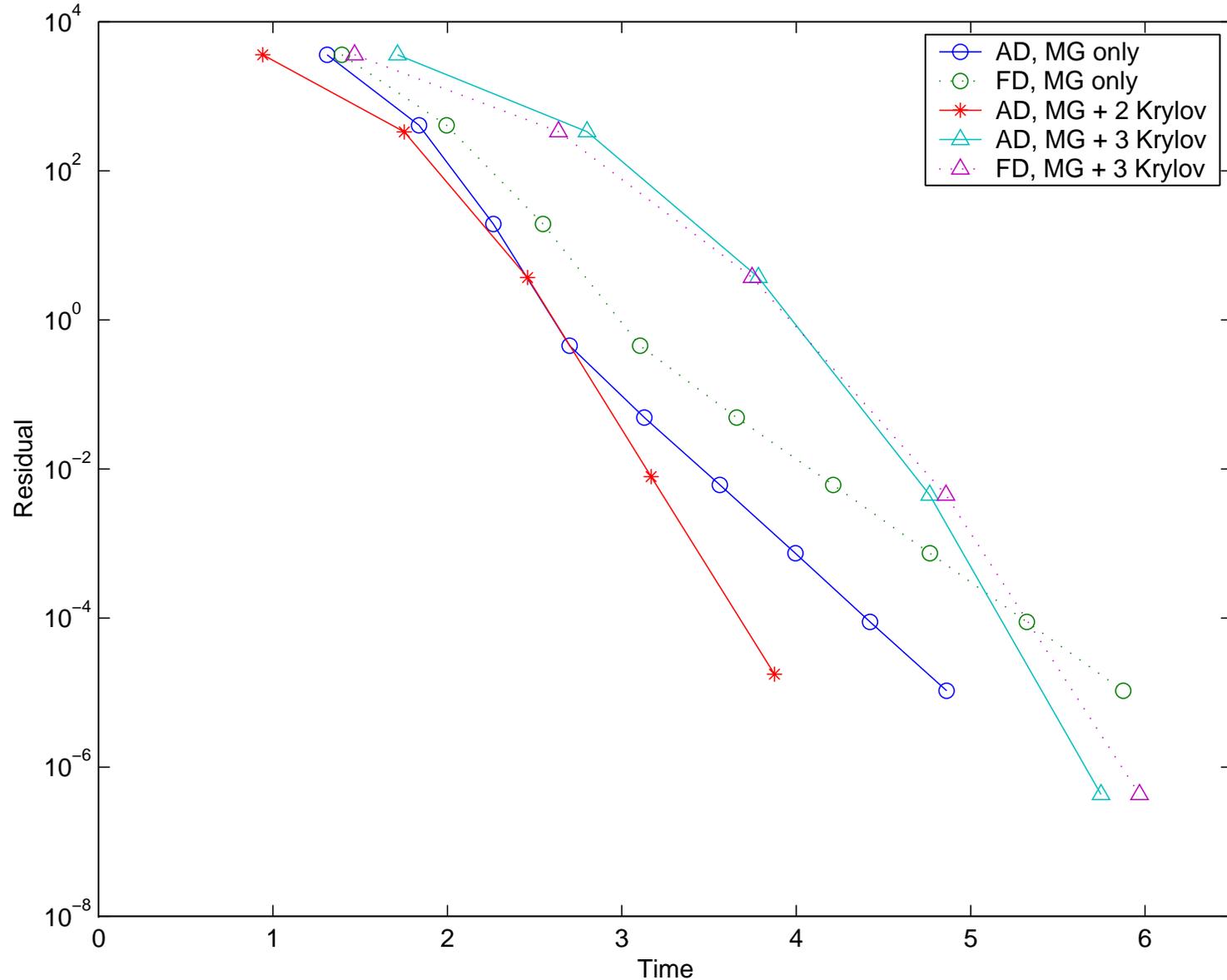
Completely Matrix-free: Iterations



Completely Matrix-free: Time



Full Multigrid: Time



Related Work

- Challenge: differentiating user-defined structures

```
int Function(SNES, Vec, Vec, void *);
```

```
typedef struct {  
  double      param;          /* test problem parameter */  
  int         mx,my;          /* discretization in x, y directions */  
  Vec        localX,localF;   /* ghosted local vectors */  
  DA         da;             /* distributed array data structure */  
  int        rank;           /* processor rank */  
} AppCtx;
```

- Software for PDE-constrained optimization
 - LNKS, multigrid, ...
- Network accessible AD software

Making AD Easier: ADIC Server



The screenshot shows a Microsoft Internet Explorer browser window titled "ADIC Application Server - Microsoft Internet Explorer". The address bar displays "http://www-unix.mcs.anl.gov/autodiff/adicserver/welcome.cgi". The main content area has a green header with "ADIC Application Server" and navigation links: "[Home] [Help] [Feedback] [Log Out]". A personalized message reads "Welcome back, Boyanal".

The interface is divided into several sections:

- Viewing options:** Includes checkboxes for "Hide AD files" and "Hide non-AD files", with "select all" and "unselect all" buttons below.
- Files:** A list showing "func.c" and "init.c".
- ADIC options:** Includes radio buttons for "Verbose mode" (selected) and "Silent mode", checkboxes for "No header file" and "No special functions" (checked), and a "Options help" link.
- Module Selection:** A "Select module:" dropdown menu currently set to "Gradient".
- Macros:** A text input field containing "-DN=20".
- Action Buttons:** A row of buttons: "run adic", "view", "delete", "rename", "copy", and "download".
- File upload:** A section titled "File upload:" with three input fields, each with a "Browse..." button, and an "upload" button at the bottom.

The browser's status bar at the bottom shows "Internet".

Future Work

- Unstructured meshes: apply AD at element function level or subdomain level (or both)?
- “Cross-country preaccumulation” at the basic block (element or vertex function) level
- Reverse mode to provide efficient $J^T v$ products (important in PDE-constrained optimization)
- Hessians
- Improved coloring
- Increased ease-of-use, documentation
- Support for variable precision, variable fidelity

What's Next

- PETSc 3.0
 - Tightly integrated AD capabilities
 - Differentiation at the element level
- QMR
 - Implementation by Bücker
 - Simultaneous J_v and $J^T w$
- Preconditioning
 - Single precision
 - Lossy compression (Gebremedhin, Manne, Pothen)
- Tensor methods using true second derivatives instead of secant approximation ($F''vw$ is cheap)
- Improved AD algorithms

Conclusions

- AD is automatable when enough information is available
- Much work remains to be done
- For more information:
 - AD
 - Griewank, *Evaluating Derivatives: Principles and Techniques of Algorithmic Differentiation*, SIAM.
 - <http://www.mcs.anl.gov/autodiff/> or <http://www.autodiff.org>
 - Tools
 - ADIFOR: <http://www.mcs.anl.gov/adifor/>
 - ADIC: <http://www.mcs.anl.gov/adic/>
 - Other AD Tools: http://www.mcs.anl.gov/autodiff/AD_Tools/
 - E-mail: hovland@mcs.anl.gov

Acknowledgments

- Jason Abate, Lucas Roh – Hostway
- Lois Curfman McInnes, Steve Benson, Lori Freitag – Argonne National Laboratory
- Christian Bischof – Aachen University of Technology
- Alan Carle, Mike Fagan – Rice University
- Po-Ting Wu – Sun Microsystems

- Generous support from DOE, NASA, NSF